Project4——生产者消费者问题

一、实验内容

有限缓冲的生产者消费者问题，生产者和消费者互斥地访问缓冲区，缓冲区满的时候，生产者线程必须等待，当缓冲区为空时，消费者线程必须等待。主要使用信号量来实现同步。

二、实验环境

虚拟机上安装 Ubuntu 15.04 64 位。

三、实验具体实现

事实上，此次实验的大体框架都在书上给出了，主要使用三个信号量 mutex, empty, full。mutex 是用来使生产者和消费者能够互斥地访问缓冲区，empty 用来检查有多少空位，full 用来检查有多少满位。main 函数中初始化缓冲，创建生产者线程和消费者线程，接着 sleep 一段时间然后退出程序，在这 sleep 期间让工作线程充分进行。主要编程工作在于生产者线程和消费者线程的实现。

生产者先检查是否有空位（使用 empty），再进行互斥地访问（使用 mutex），操作结束后退出互斥的访问，并在缓冲区增加一个满位。

```
void *producer(void *param)
{
    buffer_item ran;
    int id = *(int *) param;

    while(1)
    {
        sleep(rand() % 5);
        ran = rand() % 1000;
        sem_wait(&empty);
        sem_wait(&mutex);

        buffer[in] = ran;
        ++counter1;

        printf("producer %d produced %d\tto buffer[%d]  %d\n", id, ran, in, counter1);

        in = (in + 1) % BUFFER_SIZE;
        sem_post(&mutex);
        sem_post(&full);

    }
    return NULL;

}
```

消费者的实现与生产者相对应。

```
void *consumer(void *param)
{
    buffer_item ran;
    int id = *(int *) param;

    while(1)
    {
        sleep(rand() % 5);

        sem_wait(&full);
        sem_wait(&mutex);
        ran = buffer[out];

        ++counter2;

        printf("consumer %d consumed %d\tin buffer[%d]  %d\n", id, ran, out, counter2);

        out = (out + 1) % BUFFER_SIZE;
        sem_post(&mutex);
        sem_post(&empty);

    }
    return NULL;

}
```

在 main 函数中创建线程

```
    //create pthreads
    for(i = 0; i < pNum; ++i)
    {
        pOrder[i] = 1 + i;
        ret = pthread_create(&p_thread[i], NULL, producer, &pOrder[i]);
        if(ret != 0)
            callError();
    }
    for(i = 0; i < cNum; ++i)
    {
        cOrder[i] = 1 + i;
        ret = pthread_create(&c_thread[i], NULL, consumer, &cOrder[i]);
        if(ret != 0)
            callError();
    }
```

四、实验结果的呈现

```
gao@gao-virtual-machine:~/Desktop$ ./a.out 6 9 3
producer 3 produced 386 to buffer[0]  1
producer 4 produced 793 to buffer[1]  2
producer 2 produced 426 to buffer[2]  3
producer 8 produced 59  to buffer[3]  4
producer 6 produced 211 to buffer[4]  5
consumer 1 consumed 386 in buffer[0]  1
producer 8 produced 567 to buffer[0]  6
consumer 2 consumed 793 in buffer[1]  2
producer 4 produced 782 to buffer[1]  7
consumer 3 consumed 426 in buffer[2]  3
producer 8 produced 862 to buffer[2]  8
consumer 2 consumed 59  in buffer[3]  4
producer 5 produced 123 to buffer[3]  9
consumer 1 consumed 211 in buffer[4]  5
producer 9 produced 67  to buffer[4]  10
```

```
gao@gao-virtual-machine:~/Desktop$ ./a.out 7 4 3
producer 4 produced 793 to buffer[0]  1
producer 4 produced 386 to buffer[1]  2
consumer 2 consumed 793 in buffer[0]  1
producer 2 produced 690 to buffer[2]  3
consumer 1 consumed 386 in buffer[1]  2
producer 1 produced 926 to buffer[3]  4
producer 1 produced 426 to buffer[4]  5
producer 4 produced 736 to buffer[0]  6
consumer 3 consumed 690 in buffer[2]  3
consumer 2 consumed 926 in buffer[3]  4
producer 4 produced 567 to buffer[1]  7
producer 3 produced 530 to buffer[2]  8
producer 1 produced 862 to buffer[3]  9
consumer 1 consumed 426 in buffer[4]  5
producer 2 produced 135 to buffer[4]  10
consumer 3 consumed 736 in buffer[0]  6
producer 4 produced 929 to buffer[0]  11
```

使用信号量，可以发现生产者和消费者对缓冲区的访问不是乱序的。

五、实验的思考与讨论

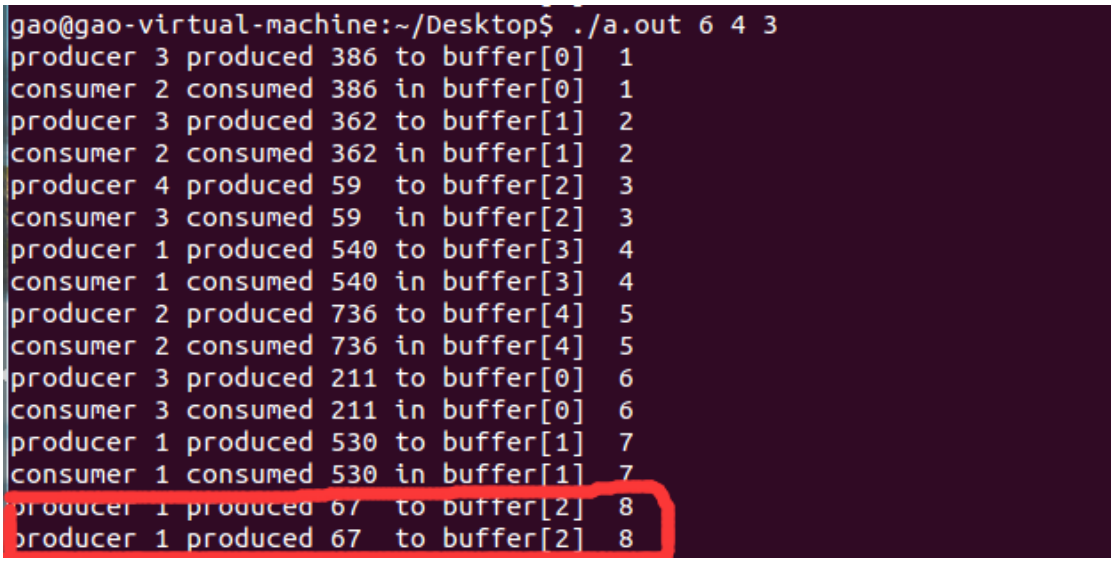
由于书上的指导比较详细，所以本次实验理解上不难，同时理解了关于信号量的几个函数的使用以后，还是比较容易写出整个程序。

但是遇到了一个 bug，一直没能解决（询问助教，助教也不是很清楚其中的缘由）。于是决定还是在报告这里写出来。

由于为了让生产者和消费者对于缓冲区的操作更加清晰一些，对于每个生产者消费者进行编号，并标明 buffer 数组的下标，这样一来比较容易看出对于

缓冲区具体的操作，容易进行测试。还加上 counter 变量用来计数生产者和消费者的数量。

在多次测试过程中就出现了一个 bug，如下图所示



最后打印出了两条一模一样的内容，思考是否是信号量同步的问题，仔细检查觉得没问题。

本来按照书上是写了 insert 和 producer 两个函数（还有 remove 和 consumer），思考是否会是函数调用中出现问题，于是干脆只写一个函数 producer（以及 consumer），但是问题依然存在。

这时，我询问了助教，认为很可能是 stdout 缓冲区的问题，于是依照实验二加上 fflush 语句强制刷新缓冲区，依然不行。这时候干脆掘弃 printf，使用 c++ 的流输出 cout，问题还是没有得到解决。

所以，可能真的是按照助教说的，有可能是 linux 内核的 bug，但也不排除我的程序还是有些问题。比较遗憾的是，这个问题最终还是没能得到解决。

六、实验总结

本次实验学习到了信号量的使用，通过生产者消费者问题的练习，对于进程同步的问题有了更加深刻的认识。不过还是很遗憾，有一个小问题最终还是没能解决，但通过对这个问题的探究，我也学到了许多。